Ridge Regression for spotify playlist

Marta Kołodziej V10102

July 2023

INTRODUCTION

The main aim of this project, was to build ridge regression for the dataset, in order to predict the popularity of the songs. The dataset contained 114 000 observations, that characterized all kinds of music, and 15 variables. With the aim of creating a ridge regression, one-hot encoding was used on variables that weren't numerical. Ridge regression was performed from the scratch and then using sklearn package cross validation with 5 k folds was implemented.

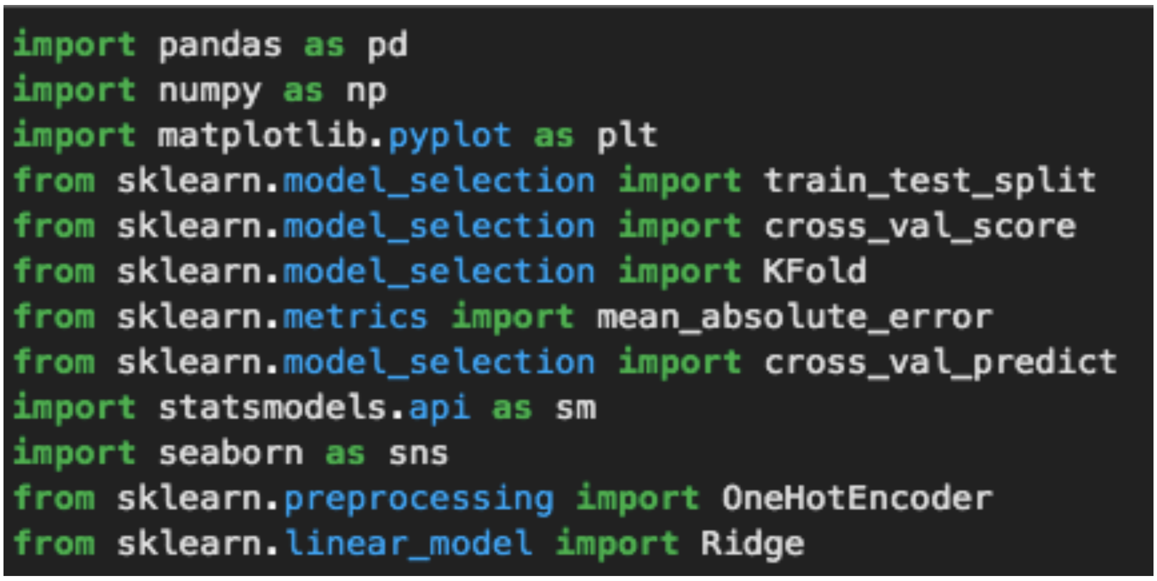
THEORY

Ridge Regression is used when model deals with multicollinearity (Taboga, 2021). Dataset includes too many variables with strong coefficients, that undermines the real power and statistical significancy of the variable and can lead to overfitting. Ridge regression adds to a model a penalty, that puts a similar constraint on the coefficients (Browniee, 2020). Penalty, regularizes the regression coefficients by shrinking them towards zero. The estimator is biased and has low variance. In ridge regression there is also used a tunning parameter alpha, that controls the impact of the shrinkage penalty on the created model (Gareth, J etc., 2013).

Cross validation using K-folds is used to evaluate the created prediction model (Nellihela, 2022). Dataset is divided into a k subset and then trained k times, while using a different fold for validation every time. The final value of the model is obtained by averaging performance metrics from every fold. In a projects K folds cross validation was used to obtain test Mean Square error to estimate the liability of the model on an unseen data. MSE measures how close the ridge regression falls from the set of data points. The lower the value, the better the model.

RIDGE REGRESSION FOR NUMERICAL VARIABLES

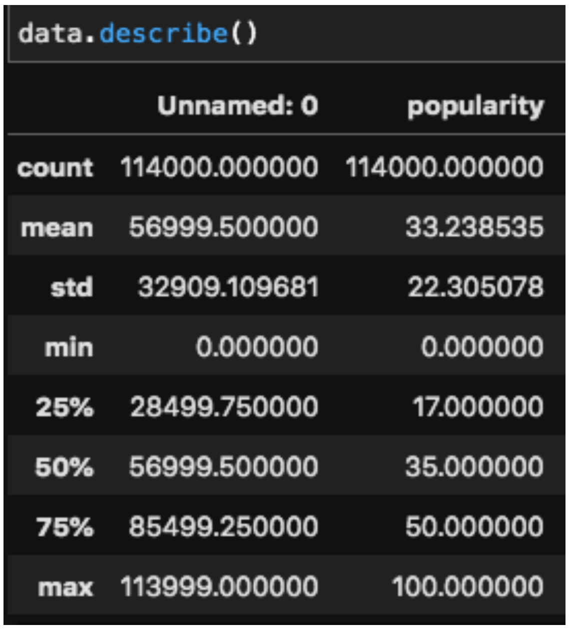
First step that was taken while writing a code was to import all the needed libraries that are displayed in a figure 1.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_predict
import statsmodels.api as sm
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import Ridge
```

Figure 1 Libraries.

Figure 2 represents basic statistics for the dependant variable. The second column shows that the mean of the popularity for all the songs in a data set equals 33.24. Due to standard deviation being 22.31, the dispersion of the data, comparing it to mean, is big. Some of the values reach the lowest (0 popularity) and the highest (100 popularity) value on a scale. The first quartile equals 17, median is 35 and the third quartile is 50.

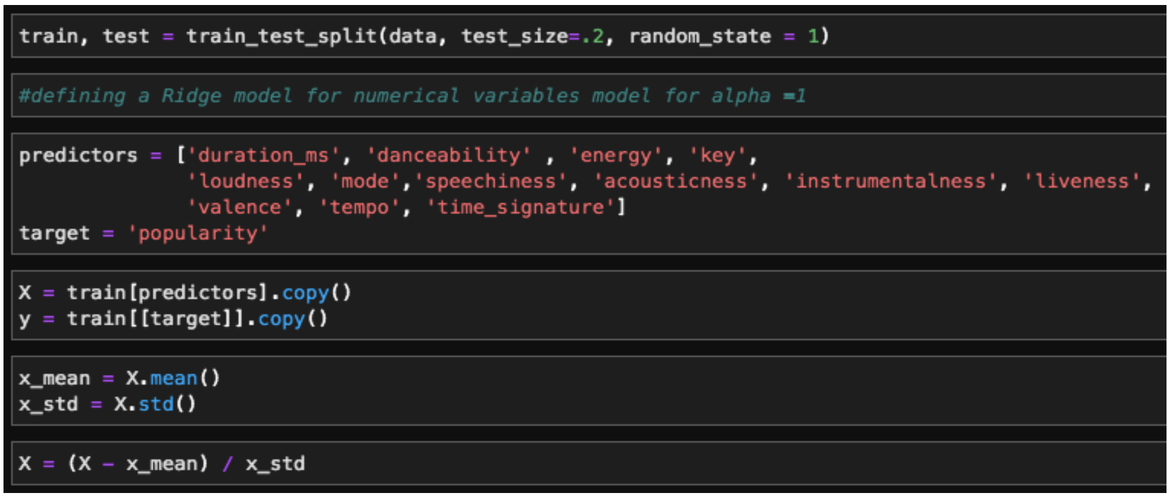


```
data.describe()
```

	Unnamed: 0	popularity
count	114000.000000	114000.000000
mean	56999.500000	33.238535
std	32909.109681	22.305078
min	0.000000	0.000000
25%	28499.750000	17.000000
50%	56999.500000	35.000000
75%	85499.250000	50.000000
max	113999.000000	100.000000

Figure 2 Basic statistics for dependant variable.

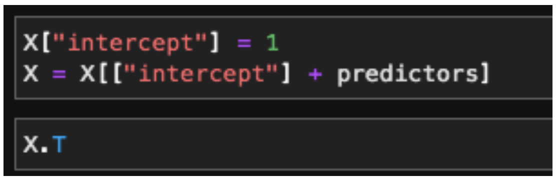
The first model that was built contained only numerical variables. The data was split into a training set and a test set (Figure 3). The next step that was taken was standardization of the data.



```
train, test = train_test_split(data, test_size=.2, random_state = 1)

#defining a Ridge model for numerical variables model for alpha =1

predictors = ['duration_ms', 'danceability' , 'energy', 'key',
              'loudness', 'mode','speechiness', 'acousticness', 'instrumentalness', 'liveness',
              'valence', 'tempo', 'time_signature']
target = 'popularity'

X = train[predictors].copy()
y = train[[target]].copy()

x_mean = X.mean()
x_std = X.std()

X = (X - x_mean) / x_std
```

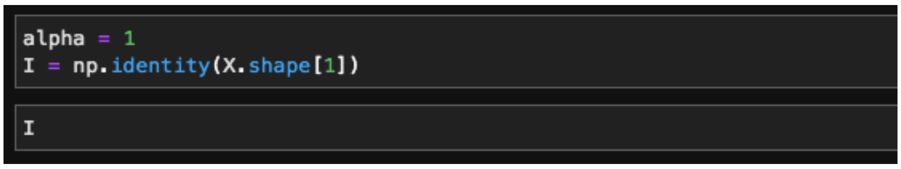
Figure 3 Data standarization..

While creating a ridge regression from a scratch the matrix of the predictors was created (Figure 4). Figure 5 presents generating identity matrix and parameter alpha that equals 1. Penalty in ridge regression is parameter alpha multiplied by the identity matrix, which is shown on a figure 6.



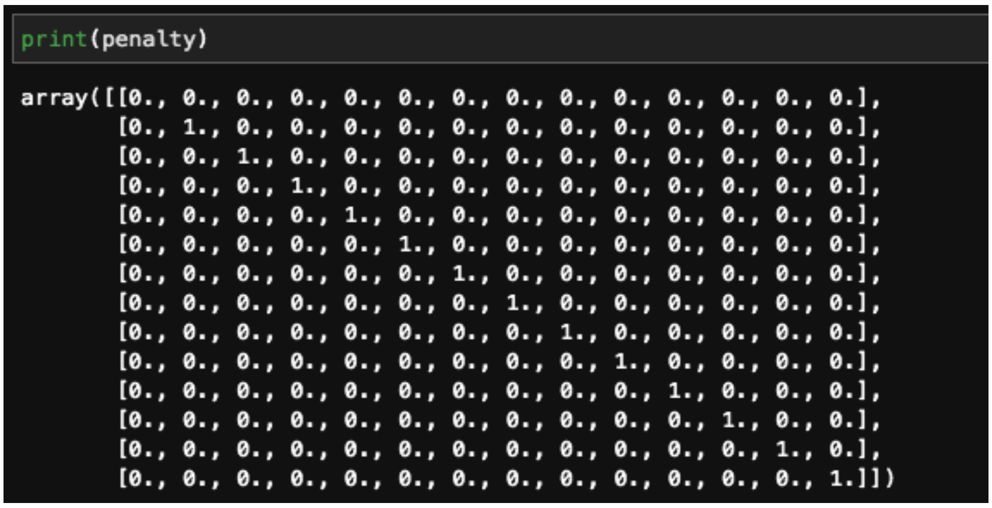
```
X["intercept"] = 1
X = X[["intercept"] + predictors]

X.T
```

Figure 4 Biulding the matrix.



```
alpha = 1
I = np.identity(X.shape[1])

I
```

Figure 5 Bulidning the identity matrix.



```
print(penalty)

array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

Figure 6 Penalty

The rigde regression was created, by using a code displayed on a figure 7. Figure 8 presents a final outcome of the final ridge regression formula, that is also displayed below:

$$y = -0{,}24x_1 + 1{,}54x_2 - 0{,}77x_3 - 0{,}09x_4 + 0{,}63x_5 - 0{,}39x_6 - 1{,}31x_7 - 0{,}26x_8 - 2{,}53x_9 + 0{,}33x_{10} - 2{,}49x_{11} + 0{,}42x_{12} + 0{,}46x_{13} + 33{,}24$$

where:

x_1- duration_ms

x_2- danceabilty

x_3- energy

x_4- key

x_5- loudness

x_6- mode

x_7- speechiness

x_8- acuosticness

x_9- instrumentalness

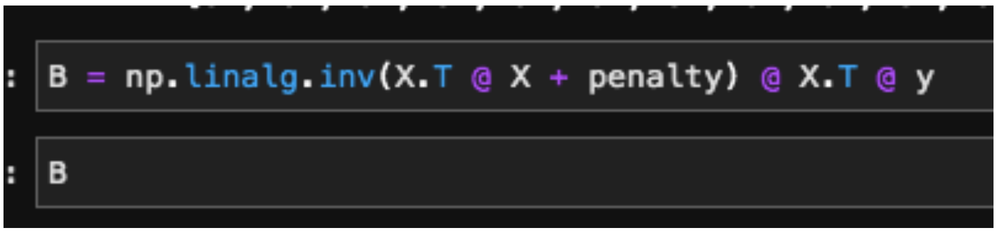
x_{10}- liveness

x_{11}- valence

x_{12}- tempo

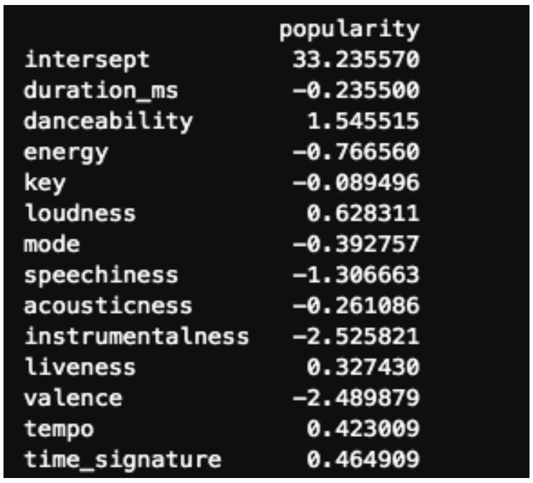
x_{13}- time_signature

In a model there are 6 positive coefficients and 7 negative ones. The variables that have the most impact on a popularity of a song are instrumentalness and valence. The more song is instrumental, the lower are the chances of getting more popular. If a message behind a song is positive, the higher the chances of it being more popular. The independent variable with the least impact on popularity is key. The scale of a song doesn't significantly change the likeness of it to score higher on charts.



```
: B = np.linalg.inv(X.T @ X + penalty) @ X.T @ y

: B
```

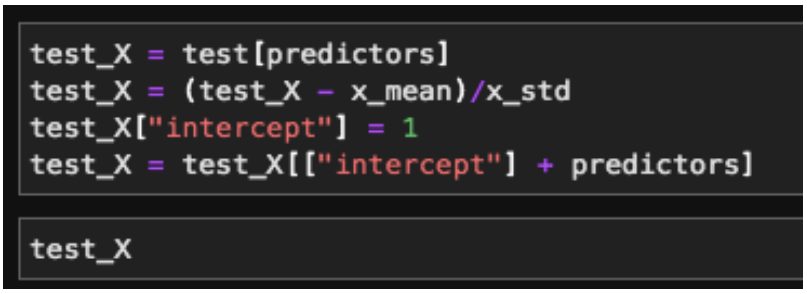
Figure 7 Final code for ridge regression.



```
                    popularity
intersept            33.235570
duration_ms          -0.235500
danceability          1.545515
energy               -0.766560
key                  -0.089496
loudness              0.628311
mode                 -0.392757
speechiness          -1.306663
acousticness         -0.261086
instrumentalness     -2.525821
liveness              0.327430
valence              -2.489879
tempo                 0.423009
time_signature        0.464909
```

Figure 8 Ridge regression for numerical values (model 1).

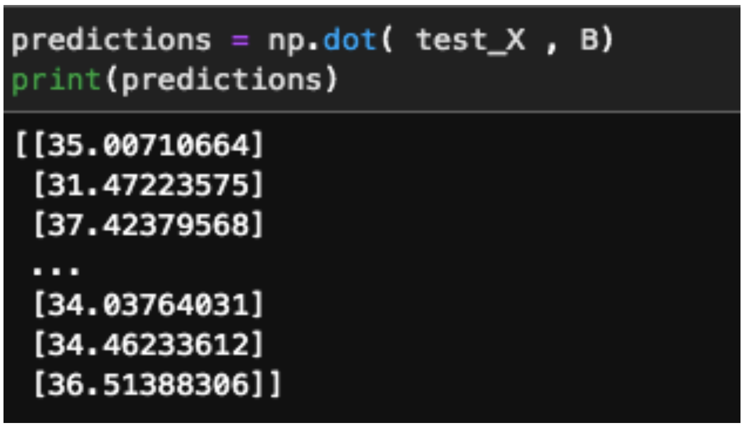
Figure 9 presents the code for calculating the predictions. The observations in a test set are multiplied by the created regression. Figure 10 presents the final predictions for model with alpha equalled 1.



```
test_X = test[predictors]
test_X = (test_X - x_mean)/x_std
test_X["intercept"] = 1
test_X = test_X[["intercept"] + predictors]
```

```
test_X
```

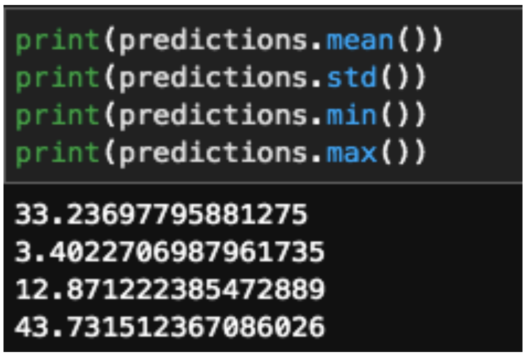
Figure 9 Prepraring the test set to predict.



```
predictions = np.dot( test_X , B)
print(predictions)
```

```
[[35.00710664]
 [31.47223575]
 [37.42379568]
 ...
 [34.03764031]
 [34.46233612]
 [36.51388306]]
```

Figure 10 Predicitons for model 1.



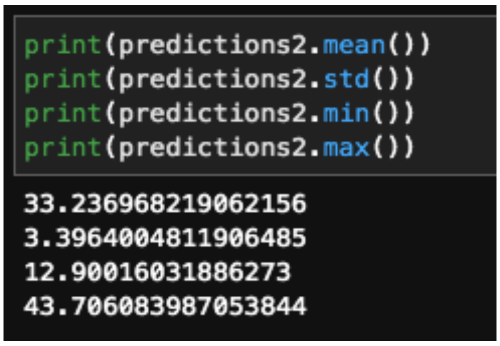
```
print(predictions.mean())
print(predictions.std())
print(predictions.min())
print(predictions.max())
```

```
33.23697795881275
3.4022706987961735
12.871222385472889
43.731512367086026
```

Figure 11 Basic statistics for model 1 predctions.

Figure 11 presents basic statistics for the predicted values. Mean is very similar to original data, but standard deviation is much smaller. The difference between maximum and minimum is also very small.

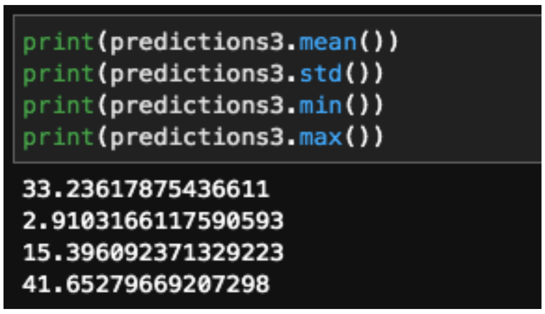
The next step was to create models with different parameter to later compare and to obtain the model with the smallest error. In the project, models with parameters 100 (figure 12), 10 000 (figure 13) and 1 000 000 (figure 14) were created. The biggest change in values happens for model 4 (alpha = 1 000 000). The maximum and minimum value are very similar to the mean. The standard deviation is minimal.



```
print(predictions2.mean())
print(predictions2.std())
print(predictions2.min())
print(predictions2.max())
```

```
33.236968219062156
3.3964004811906485
12.90016031886273
43.706083987053844
```

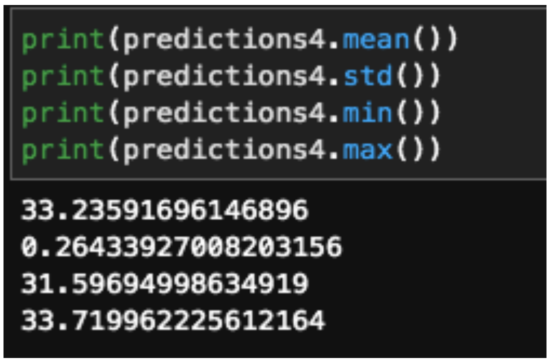
Figure 12 Basic statistics for model 2 predctions.



```
print(predictions3.mean())
print(predictions3.std())
print(predictions3.min())
print(predictions3.max())
```

```
33.23617875436611
2.9103166117590593
15.396092371329223
41.65279669207298
```

Figure 13 Basic statistics for model 3 predctions.

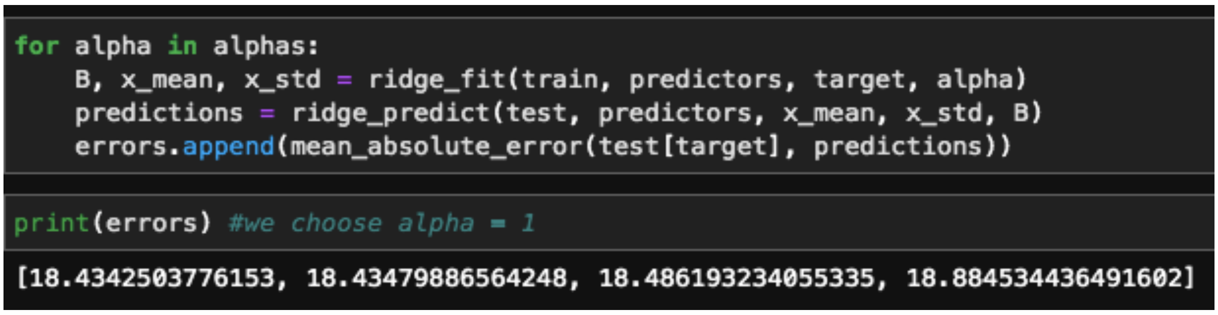


```
print(predictions4.mean())
print(predictions4.std())
print(predictions4.min())
print(predictions4.max())
```

```
33.23591696146896
0.26433927008203156
31.59694998634919
33.719962225612164
```

Figure 14 Basic statistics for model 4 predctions.

For every model built an error was calculated and displayed in a figure 15. The errors are also very similar, but the smallest value is reached for parameter equalled 1, so the first ridge regression should be chosen to properly predict popularity of the songs.



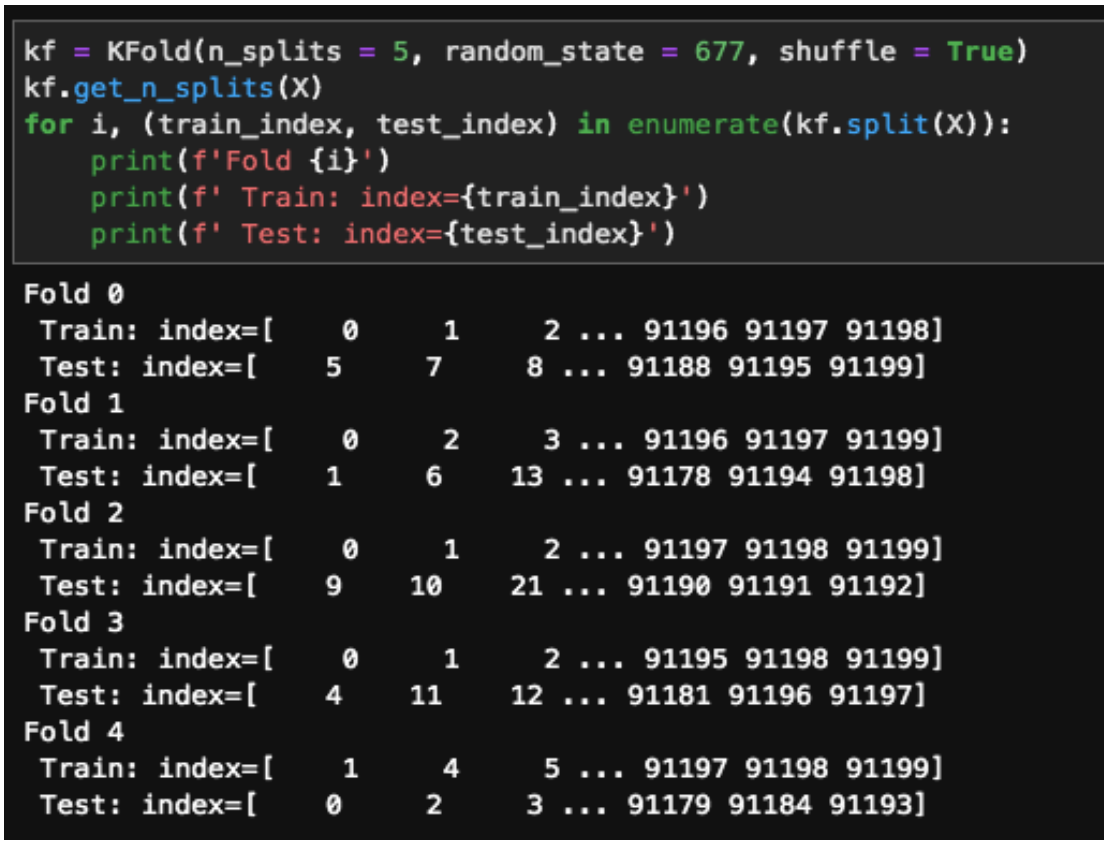
```
for alpha in alphas:
    B, x_mean, x_std = ridge_fit(train, predictors, target, alpha)
    predictions = ridge_predict(test, predictors, x_mean, x_std, B)
    errors.append(mean_absolute_error(test[target], predictions))
```

```
print(errors) #we choose alpha = 1
```

```
[18.4342503776153, 18.43479886564248, 18.486193234055335, 18.884534436491602]
```

Figure 15 Errors for every model built

The last step for ridge regression with numerical variables was cross validation using 5 folds. Figure 16 presents created folds, that the data will be tested and trained on. The value of a mean squared error of the cross validation with 5 folds is 22,04.



```
kf = KFold(n_splits = 5, random_state = 677, shuffle = True)
kf.get_n_splits(X)
for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f'Fold {i}')
    print(f' Train: index={train_index}')
    print(f' Test: index={test_index}')
```

```
Fold 0
 Train: index=[    0     1     2 ... 91196 91197 91198]
 Test: index=[    5     7     8 ... 91188 91195 91199]
Fold 1
 Train: index=[    0     2     3 ... 91196 91197 91199]
 Test: index=[    1     6    13 ... 91178 91194 91198]
Fold 2
 Train: index=[    0     1     2 ... 91197 91198 91199]
 Test: index=[    9    10    21 ... 91190 91191 91192]
Fold 3
 Train: index=[    0     1     2 ... 91195 91198 91199]
 Test: index=[    4    11    12 ... 91181 91196 91197]
Fold 4
 Train: index=[    1     4     5 ... 91197 91198 91199]
 Test: index=[    0     2     3 ... 91179 91184 91193]
```

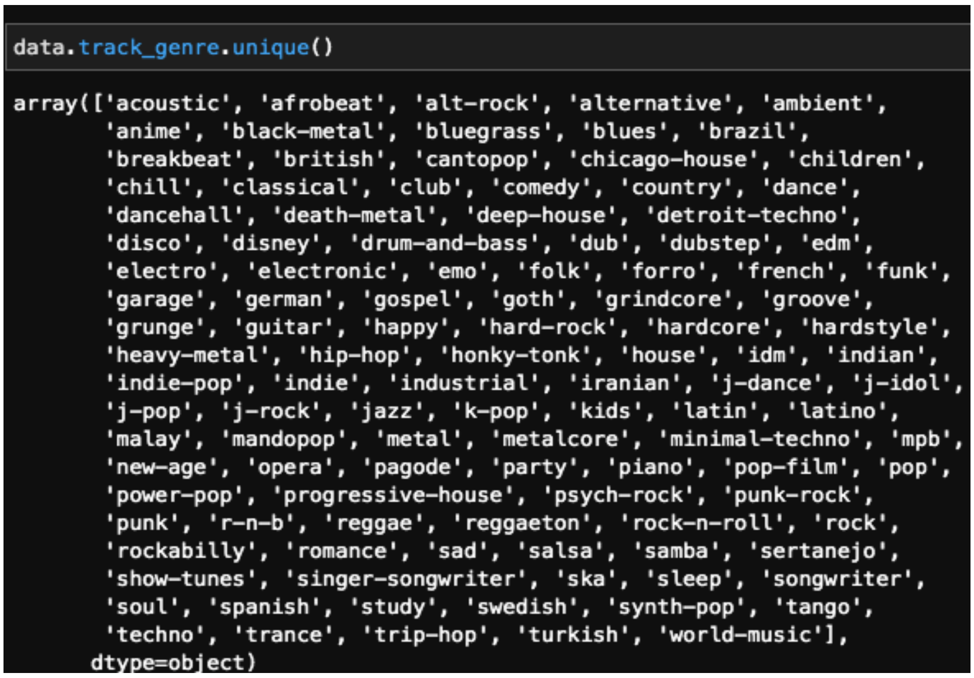
Figure 16 Cross Validation using 5 folds.

SECOND MODEL

In a dataset, there are also two non-numerical variables. Track_genre is an object- every genre is described (Figure 17), while explicit is a Boolean-true or false explicitness of a song (Figure 18).

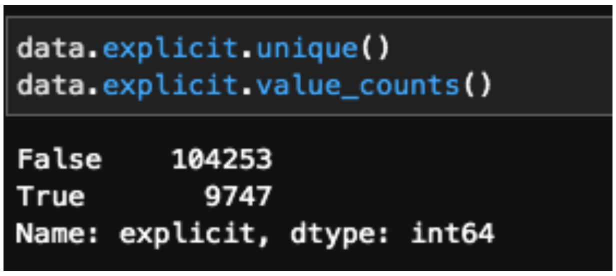
RIDGE REGRESSION FOR ALL VARIABLES

In a dataset, there are also two non-numerical variables. Track_genre is an object- every genre is described (Figure 17), while explicit is a Boolean-true or false explicitness of a song (Figure 18).



```
data.track_genre.unique()

array(['acoustic', 'afrobeat', 'alt-rock', 'alternative', 'ambient',
       'anime', 'black-metal', 'bluegrass', 'blues', 'brazil',
       'breakbeat', 'british', 'cantopop', 'chicago-house', 'children',
       'chill', 'classical', 'club', 'comedy', 'country', 'dance',
       'dancehall', 'death-metal', 'deep-house', 'detroit-techno',
       'disco', 'disney', 'drum-and-bass', 'dub', 'dubstep', 'edm',
       'electro', 'electronic', 'emo', 'folk', 'forro', 'french', 'funk',
       'garage', 'german', 'gospel', 'goth', 'grindcore', 'groove',
       'grunge', 'guitar', 'happy', 'hard-rock', 'hardcore', 'hardstyle',
       'heavy-metal', 'hip-hop', 'honky-tonk', 'house', 'idm', 'indian',
       'indie-pop', 'indie', 'industrial', 'iranian', 'j-dance', 'j-idol',
       'j-pop', 'j-rock', 'jazz', 'k-pop', 'kids', 'latin', 'latino',
       'malay', 'mandopop', 'metal', 'metalcore', 'minimal-techno', 'mpb',
       'new-age', 'opera', 'pagode', 'party', 'piano', 'pop-film', 'pop',
       'power-pop', 'progressive-house', 'psych-rock', 'punk-rock',
       'punk', 'r-n-b', 'reggae', 'reggaeton', 'rock-n-roll', 'rock',
       'rockabilly', 'romance', 'sad', 'salsa', 'samba', 'sertanejo',
       'show-tunes', 'singer-songwriter', 'ska', 'sleep', 'songwriter',
       'soul', 'spanish', 'study', 'swedish', 'synth-pop', 'tango',
       'techno', 'trance', 'trip-hop', 'turkish', 'world-music'],
      dtype=object)
```

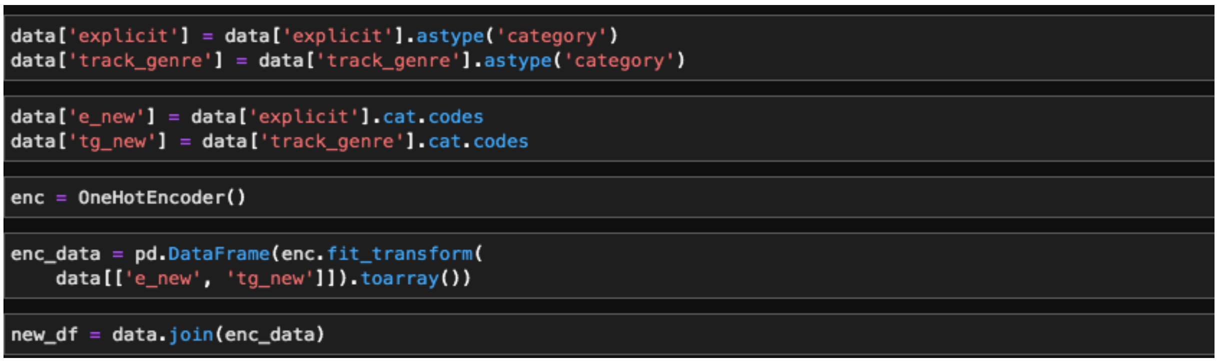
Figure 17 Values for variable track_genre.



```
data.explicit.unique()
data.explicit.value_counts()

False    104253
True       9747
Name: explicit, dtype: int64
```

Figure 18 Values for variable explicit.

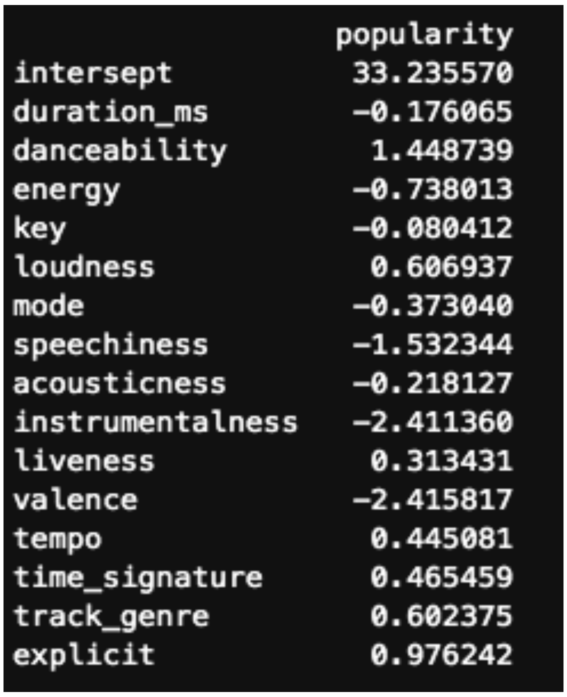
Figure 19 presents code for one-hot encoding in order to change both variables into a numerical variable. After that, new dataset was created to build ridge regression that contains all the variables.



```
data['explicit'] = data['explicit'].astype('category')
data['track_genre'] = data['track_genre'].astype('category')

data['e_new'] = data['explicit'].cat.codes
data['tg_new'] = data['track_genre'].cat.codes

enc = OneHotEncoder()

enc_data = pd.DataFrame(enc.fit_transform(
    data[['e_new', 'tg_new']]).toarray())

new_df = data.join(enc_data)
```

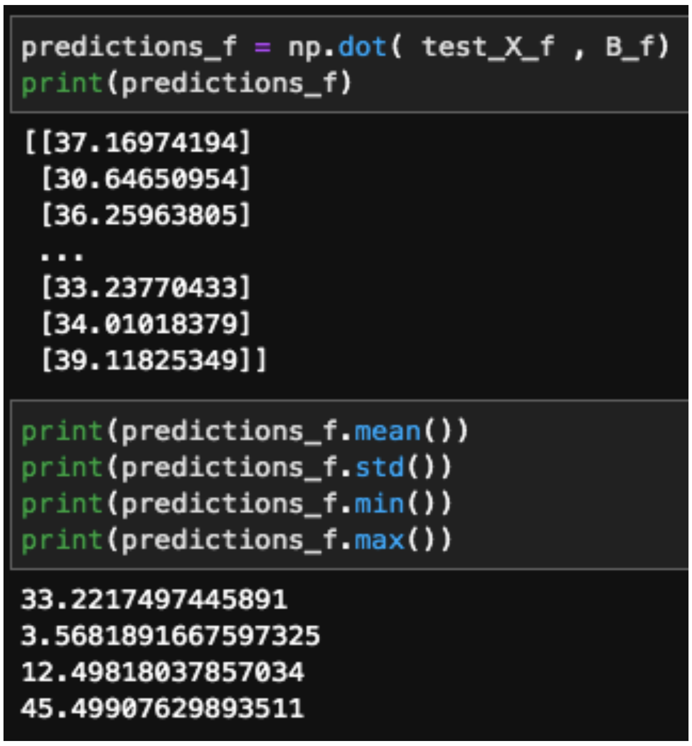
Figure 19 One hot encoding.

Ridge regression for all the variables was built cognately as the first model. Figure 20 presents the coefficiency of all the variables. The most impactful variables are valence and instrumentalness. Variable key is still the one with the least impact on the popularity of the song.



```
                      popularity
intersept              33.235570
duration_ms            -0.176065
danceability            1.448739
energy                 -0.738013
key                    -0.080412
loudness                0.606937
mode                   -0.373040
speechiness            -1.532344
acousticness           -0.218127
instrumentalness       -2.411360
liveness                0.313431
valence                -2.415817
tempo                   0.445081
time_signature          0.465459
track_genre             0.602375
explicit                0.976242
```

Figure 20 Ridge regression fo all the variables (model 5)



```
predictions_f = np.dot( test_X_f , B_f)
print(predictions_f)
```

```
[[37.16974194]
 [30.64650954]
 [36.25963805]
 ...
 [33.23770433]
 [34.01018379]
 [39.11825349]]
```

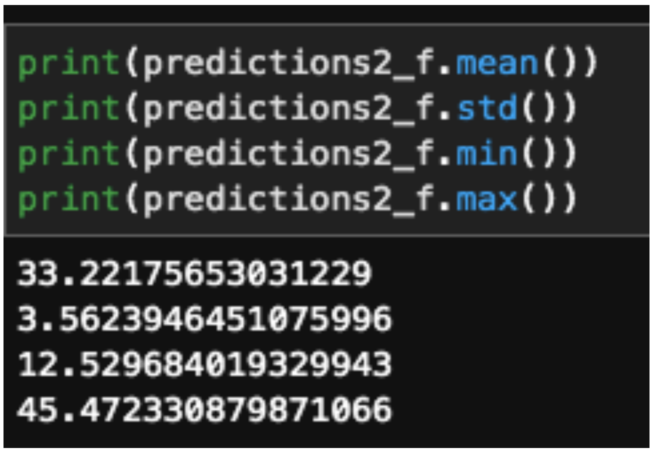
```
print(predictions_f.mean())
print(predictions_f.std())
print(predictions_f.min())
print(predictions_f.max())
```

```
33.2217497445891
3.5681891667597325
12.49818037857034
45.49907629893511
```

Figure 21 Predictions for model 5.

Predictions for this model are presented in a figure 21. The mean popularity is 33.22, standard deviation 3.57. Maximal popularity of the song is 45.5 and minimum 12.5. Predictions does not differ that much for a model with a parameter 100 (figure 22) and parameter 10 000 (figure 23). Predictions for a model with an alpha equalled 1 000 000 changes (Figure 24). The standard deviation is minimal and the difference between the predicted popularity as well. For this regression, error is the biggest, which suggest rejecting model with a parameter 1 000 0000.

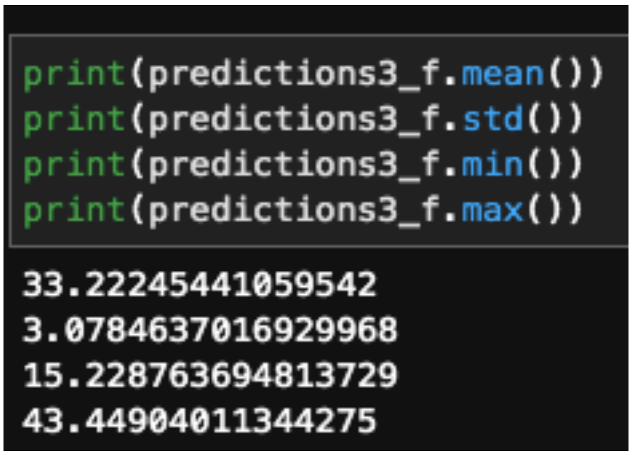
Errors for the first three regressions are similar, however the lowest value is for a regression with an alpha equalled 1.



```
print(predictions2_f.mean())
print(predictions2_f.std())
print(predictions2_f.min())
print(predictions2_f.max())
```

```
33.22175653031229
3.5623946451075996
12.529684019329943
45.472330879871066
```

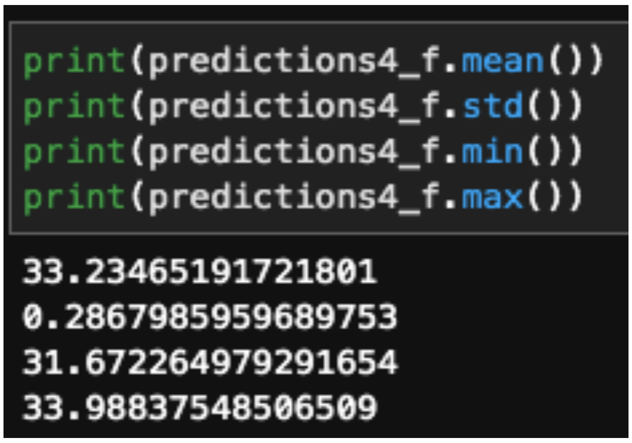
Figure 22 Basic statistics for model 6.



```
print(predictions3_f.mean())
print(predictions3_f.std())
print(predictions3_f.min())
print(predictions3_f.max())
```

```
33.22245441059542
3.0784637016929968
15.228763694813729
43.44904011344275
```

Figure 23 Basic statistics for model 7.

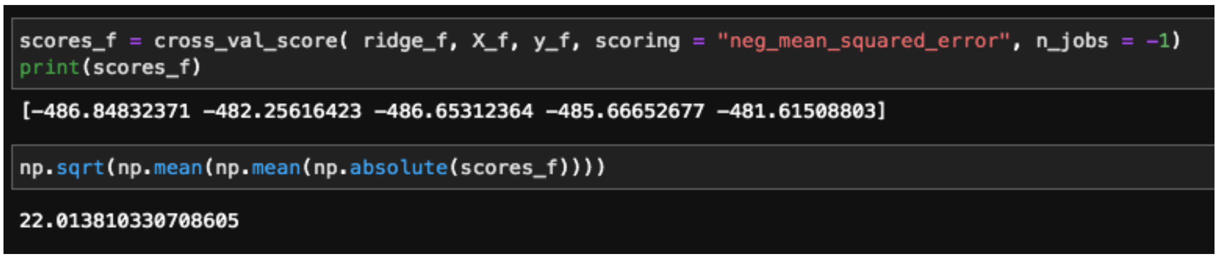


```
print(predictions4_f.mean())
print(predictions4_f.std())
print(predictions4_f.min())
print(predictions4_f.max())
```

```
33.23465191721801
0.2867985959689753
31.672264979291654
33.98837548506509
```

Figure 24 Basic statistics for model 8.

Cross validation with 5 folds was performed accordingly to the first model. Mean square error equalled 22,01, which is slightly better than the model with only numerical values.



```
scores_f = cross_val_score( ridge_f, X_f, y_f, scoring = "neg_mean_squared_error", n_jobs = -1)
print(scores_f)
```

```
[-486.84832371 -482.25616423 -486.65312364 -485.66652677 -481.61508803]
```

```
np.sqrt(np.mean(np.mean(np.absolute(scores_f))))
```

```
22.013810330708605
```

Figure 25 Errors for all the models built.

SUMMARY

Data set consisted of 114 000 observations and 15 variables. Two of them were changed into a numerical using one-hot encoding. Ridge regression was created and applied twice to different variables and parameters. Errors form each model suggested taking in consideration results from regression with parameter that equalled 1. After performing cross validation with 5 folds, mean square error suggested predicting with a model that has all the variables included in a data set. Both errors for these models were the lowest. In a model there are 7 positive coefficients and 8 negative ones. The most impactful variables are valence and instrumentalneses. The ones that also can change the popularity are speechiness and danceability. The least impactful variable is key of the song. Temp and time signature also do not have a strong significance for a dependant variable.

Source:

1. Browniee, J.(2020). *A Gentle Introduction to k-fold Cross-Validation.* Downloaded 15.07.2023r from https://machinelearningmastery.com/k-fold-cross-validation/.

2. Taboga, Marco (2021). *"Ridge regression", Lectures on probability theory and mathematical statistics*. Kindle Direct Publishing. Online appendix. https://www.statlect.com/fundamentals-of-statistics/ridge-regression.

3. Nellihela, P. (2022). *What is K-fold Cross Validation?* Downloaded on 16.07.2023 from ttps://towardsdatascience.com/what-is-k-fold-cross-validation-5a7bb241d82f.

4. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). *An introduction to statistical learning: with applications in R. Second Edition.* New York, 2013.